

On the Latency and Energy Efficiency of Erasure-Coded Cloud Storage Systems

Akshay Kumar, Ravi Tandon, T. Charles Clancy

Abstract

The increase in data storage and power consumption at data-centers has made it imperative to design energy efficient Distributed Storage Systems (DSS). The energy efficiency of DSS is strongly influenced not only by the volume of data, frequency of data access and redundancy in data storage, but also by the heterogeneity exhibited by the DSS in these dimensions. To this end, we propose and analyze the energy efficiency of a heterogeneous distributed storage system in which n storage servers (disks) store the data of R distinct classes. Data of class i is encoded using a (n, k_i) erasure code and the (random) data retrieval requests can also vary across classes. We show that the energy efficiency of such systems is closely related to the average latency and hence motivates us to study the energy efficiency via the lens of average latency. Through this connection, we show that erasure coding serves the dual purpose of reducing latency and increasing energy efficiency. We present a queuing theoretic analysis of the proposed model and establish upper and lower bounds on the average latency for each data class under various scheduling policies. Through extensive simulations, we present qualitative insights which reveal the impact of coding rate, number of servers, service distribution and number of redundant requests on the average latency and energy efficiency of the DSS.

Index Terms

Erasure Codes, Distributed Storage, Fork-Join Queues, Latency, Energy Efficiency, Multi-class queuing system.

I. INTRODUCTION

Cloud based storage systems are emerging to gain significant prominence due to their highly virtualized infrastructure that presents cost-effective and simple to use elastic network resources. The backbone infrastructure of the cloud is comprised of distributed storage systems (DSS), in which the data is stored and accessed from commodity storage disks. Coding of data across distributed disks provides fault tolerance by providing reliability against unexpected disk failures. There has been a recent paradigm shift from classical replication based codes to erasure codes because they provide higher fault tolerance at the same storage cost [2]. As a result, a number of commercial DSS such as Google Colossus, Windows Azure etc. are transitioning to the use of erasure codes [3]–[5]. Besides providing fault tolerance and minimizing storage cost, another important aspect which deserves equal, if not more attention is the *energy efficiency* of DSS.

Over the last decade, the dramatic usage of data has lead to an enormous increase in the volume of stored (archival) data and the frequency of data access to a DSS [6]. This translates to more and more servers being added to the data-center operating at higher server utilization levels. As a result, the energy consumption of the data-centers is increasing steeply and adds up to its operational cost. According to [7], energy consumed by the data centers globally has increased by 19% in 2012 and storage systems in a large data-center consume up to 40% of the total energy [8]. Hence, there is a need to devise energy efficient data storage schemes. The existing techniques for energy-efficient data storage are based on variants of schemes that involve powering off storage devices [9]–[11].

Energy efficiency is a system wide property and while some metrics focus on the energy efficiency of hardware or software components [12], others are based on the usage of physical resources (such as CPU, memory, storage etc.) by the running applications or servers. For the scope of this work, we focus on the *data transfer throughput metric* [13], which measures energy efficiency as the amount of data

A. Kumar and T. C. Clancy are with the Hume Center for National Security and Technology and Dept. of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA USA. Email: {akshay2, tcc}@vt.edu. R. Tandon is with the Discovery Analytics Center, Dept. of Computer Science, Virginia Tech, Blacksburg, VA USA. Email: tandonr@vt.edu

Parts of this work were presented at the Globecom 2014 conference [1].

processed in the DSS per unit amount of energy expended across all distributed servers. Therefore, the energy efficiency of DSS is strongly influenced by the volume of data transferred (per request), frequency of data storage/access requests, service rate of each server and the degree of redundancy in data storage.

The energy efficiency of a DSS is also closely related to its read/write latency¹. The data stored and accessed from the cloud is broadly classified into two categories [14], [15]:

- *Hot-data*: this could refer to data which is frequently accessed (i.e., a higher job request rate). Furthermore, it is desirable to provide higher redundancy/fault tolerance when storing such data.
- *Cold-data*: this could refer to data which is infrequently accessed or archival data. Such data does not necessarily mandate to be coded and stored with higher fault tolerance, as it is seldom accessed by the users.

When the data is infrequently accessed as in case of *Cold-data*, the average latency is reduced and it also improves the energy efficiency of DSS [16]. Another case in point is that increasing redundancy as in *Hot-data* improves fault-tolerance but generally results in increased latency [17]. The energy efficiency in this case decreases due to increase in power consumption as more servers are involved in processing the same data request. Thus the latency of DSS is closely tied with its energy efficiency. Therefore in this work, we study the energy efficiency of a DSS through the lens of average latency of DSS.

As mentioned earlier, the erasure coded DSS, due to their several merits over replication based codes, have gained significant prominence in recent times. Therefore, in this work we study the relationship between latency and energy efficiency for such systems. In a erasure coded DSS, the data of each user is stored across n disks (or servers) using a (n, k) optimal Maximum-Distance-Separable (MDS) code. By the property of MDS codes, accessing the data stored at *any* k out of n servers suffices to recover the entire data of a user (also referred to as successful completion of the *job request*² of that user). The processing of job requests in DSS is typically analyzed using Fork-Join (F-J) queues [19], [20]. A (n, k) F-J queue consists of n independently operating queues corresponding to each of the n servers. Every job arriving in the system is split n ways and enters the queues of all n servers simultaneously. A queuing theoretic latency analysis of the (n, k) F-J system has been done in [17] (also see [21]–[23]). The key findings of these papers is that using erasure coding and sending redundant requests (requests to more than k servers for a (n, k) F-J system) can significantly reduce the latency of a DSS.

However, most of the aforementioned literature considers a homogenous storage architecture and there is no distinction (from system's perspective) between any two job requests entering the system. However, that is hardly the case with real DSS, wherein as mentioned earlier (see *Hot-data* vs. *Cold-data*), the job requests can be classified into one of the several classes based on the job arrival rate or fault-tolerance/storage requirements. For instance, the leading cloud storage providers such as Amazon S3, Windows Azure etc. allow their customers to choose from multiple storage classes that differ in redundancy/fault-tolerance, data availability and access latency [24], [25]. They have a low redundancy storage class designed for infrequently/archival data. Therefore, motivated by this fact, we consider a $(n, k_1, k_2, \dots, k_R)$ multi-tenant DSS for R distinct data classes, a generalization of the homogenous (n, k) DSS in [17]. Data of class i ($\forall i \in \{1, 2, \dots, R\}$) is stored across n servers using a (n, k_i) erasure (MDS) code. The arrivals³ of job requests of class i are assumed to follow a Poisson distribution with rate λ_i .

The key contributions of this paper are:

- A multi-tenant DSS is proposed and analyzed through the Fork Join framework to account for the heterogeneity in job arrival rates and fault-tolerance requirements of different data classes.
- A *data throughput* based energy efficiency metric is defined for the heterogeneous DSS operating under any given scheduling policy. For the special case of single server and data class, we showed that the average latency and energy efficiency of DSS are closely related to each other. Therefore,

¹Here, latency refers to the time taken to process a data request, measured relative to the time at which it enters the DSS. For the scope of this work, we consider latency to be the sum of queuing delay and service time, and assume the other delays to be relatively negligible.

²We restrict our attention to read requests because in most of the practical DSS, such as HDFS [18], Windows Azure [5] etc. the user's data is written only once to the storage nodes but it can be retrieved multiple times by the user.

³Job arrivals refers to the time instants at which job requests enters the queues of the servers in the DSS.

using a queuing-theoretic approach, we provided lower and upper bounds on the average latency for jobs of class i ($\forall i \in \{1, 2, \dots, R\}$) in the proposed F-J framework under various scheduling policies such as First-Come-First-Serve (FCFS), preemptive and non-preemptive priority scheduling policies.

- We studied the impact of varying the code-rate on the latency, energy efficiency and network bandwidth consumed by DSS. Increasing code-rate reduces latency and increases energy efficiency. However, this comes at the cost of increased storage space and (write) bandwidth. We also obtained interesting insights from investigating the impact of varying the number of servers, heavy-tail arrival/service distributions in the DSS.
- Lastly, we studied the impact of varying the number of redundant requests (sending requests to more than k servers for (n, k) MDS code) to the DSS. We observed that sending redundant requests reduces latency and increases energy efficiency. Thus, full redundancy results in minimum latency and maximum energy efficiency for each data-class.

II. RELATED WORK

A number of good MDS codes such as LINUX RAID-6 and array codes (EVENODD codes, X-code, RDP codes) have been developed to encode the data stored on cloud (see [26] and references therein). These codes have very low encoding/decoding complexity as they avoid Galois Field arithmetic (unlike the classical Reed-Solomon MDS codes) and involve only XOR operations. However, they are usually applicable upto two or three disk failures. Also, in the event of disk failure(s), Array codes and recently introduced Regenerating codes reduce disk and network I/O respectively. Recently, non-MDS codes such as Tornado, Raptor and LRC codes [27], [28] have been developed for erasure coded storage. Although the fault-tolerance is not as good as MDS codes, they achieve higher performance due to lower repair bandwidth and I/O costs.

The latency analysis of (MDS) erasure coded (n, k) homogenous DSS has been well investigated in [17], [21], [22] which provide queuing theoretic bounds on average latency. A related line of work [23], [29] independently showed that sending requests to multiple servers always reduces the (read) latency. Then Liang et. al. [30] extended the latency analysis to a (n, k, L) DSS, in which n of a total L number of independent servers are used to store the (n, k) MDS code. It assumed a “constant+exponential” model for the service time of jobs. The authors in [31], [32] developed load-adaptive algorithms that dynamically vary job size, coding rate and number of parallel connections to improve the delay-throughput tradeoff of key-value storage systems. These solutions were extended for heterogeneous services with mixture of job sizes and coding rate. Recently, Xiang et. al. [33] provided a tight upper bound on average latency, assuming arbitrary erasure code, multiple file types and a general service time distribution. This was then used to solve a joint latency and storage cost optimization problem by optimizing over the choice of erasure code, placement of encoded chunks and the choice of scheduling policy.

Data-centers while configured for peak-service demand, end up being highly underutilized. Furthermore, the hardware components in storage systems are not power proportional, with idle mode consuming roughly 60% of that of a busy power [34]. This has resulted in significant research in designing/implementing power efficient schemes. Most of the current literature focuses on power management via performance scaling (such as DVFS [35], [36]) or low-power states [37]. Recently, Liu et. al. [38] investigated the effect of varying various system operations such as processing speed, system on/off decisions etc. on the power-delay performance from a queuing theoretic perspective. This work was extended in [39], wherein a joint speed scaling and sleep state management approach was proposed, that determines the best low-power state and frequency setting by examining the power consumption and average response time for each pair.

However, the work in [38], [39] does not present the power-delay performance analysis in a erasure coded DSS. Also it focuses on power consumption rather than the more relevant, energy efficiency of DSS. Therefore, in this work, we study the relationship between energy efficiency and average latency in a (MDS) erasure coded heterogeneous DSS for different scheduling policies.

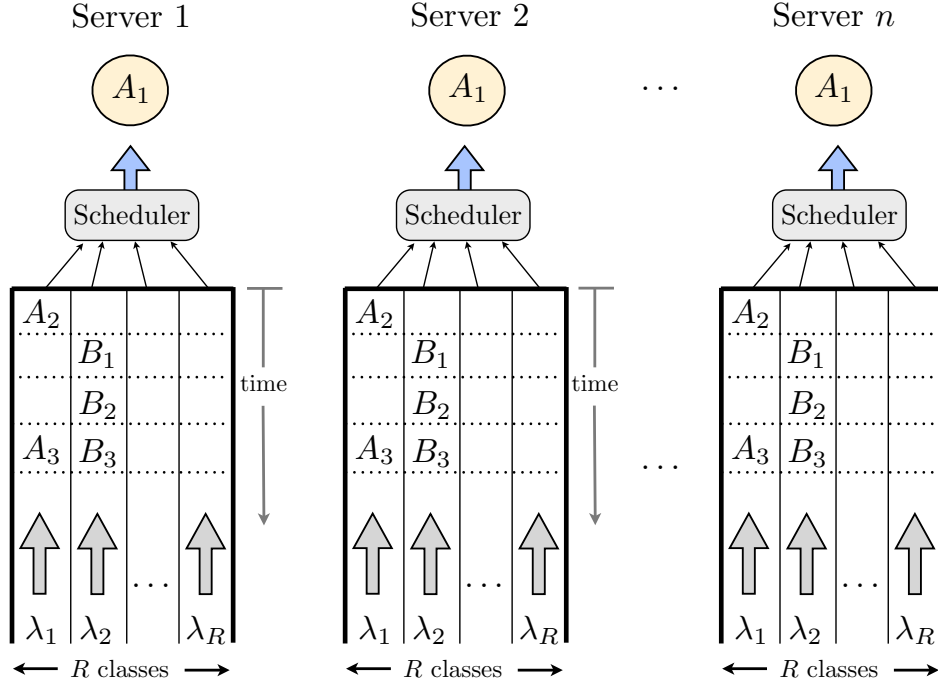


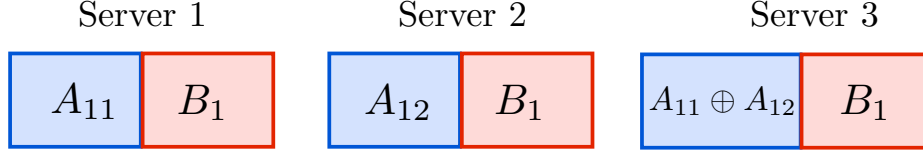
Fig. 1: System Model.

III. SYSTEM MODEL

A heterogeneous multi-tenant $(n, k_1, k_2, \dots, k_R)$ DSS (shown in Fig. 1) consists of n servers that store the data of R distinct classes. The R classes differ from each other in the fault-tolerance, storage requirements and frequency of access of the stored data. The data of class i (which is assumed to be of size l_i) is partitioned into k_i equal size fragments and then stored across n servers using a (n, k_i) Maximum-Distance-Separable (MDS) code. Thus each server stores, $1/k_i$ fraction of original data. The arrival process for request of class i is assumed to be Poisson with rate λ_i . The service time at each server is assumed to follow an exponential distribution with service rate μ (per unit file size) [32]. The effective service rate at any server for jobs of class i is $\mu_i = \frac{k_i \mu}{l_i}$ since each server stores $1/k_i$ fraction of data.

Example 1. We now present a representative example to illustrate the system model. Consider a $(n, k_1, k_2) = (3, 2, 1)$ two-class DSS. Data for the two classes A and B are encoded across $n = 3$ servers using $(3, 2)$ and $(3, 1)$ MDS codes respectively as shown in Fig. 2. Let A_1 and B_1 denote two files of class A and B respectively that need to be coded and stored across the servers. Then for the $(3, 2)$ MDS code, A_1 is split into two sub-files, A_{11} and A_{12} , of equal size and are stored on any two servers (servers 1 and 2 in Fig. 2). Then the remaining server (i.e. server 3) stores $A_{11} \oplus A_{12}$. Thus each server stores half the size of original file and the entire file can be recovered from any two servers. The $(3, 1)$ MDS code for file B_1 , is a simple replication code in which each server stores the copy of entire file of class B and thus can be recovered by accessing the data from any one server.

The evolution of system state in this example, depends on the local scheduling policy at each server. Although there exists various scheduling policies, in this work we consider First-Come-First-Serve (FCFS), preemptive and non-preemptive priority queuing policies at each server. In FCFS scheduling, all data classes are equal priority. At each server, the job that enters first in the buffer is served first. In a priority queuing policy, the data classes are assigned different priority levels. A job of a particular class will be served only when there are no outstanding jobs of classes with higher priority level. A priority queuing policy is further classified as preemptive or non-preemptive based on whether or not the job in server can be preempted by a job of higher priority level.



(3, 2) code: class A (3, 1) code: class B

Fig. 2: MDS codes for data storage in a two-class Fork-Join system.

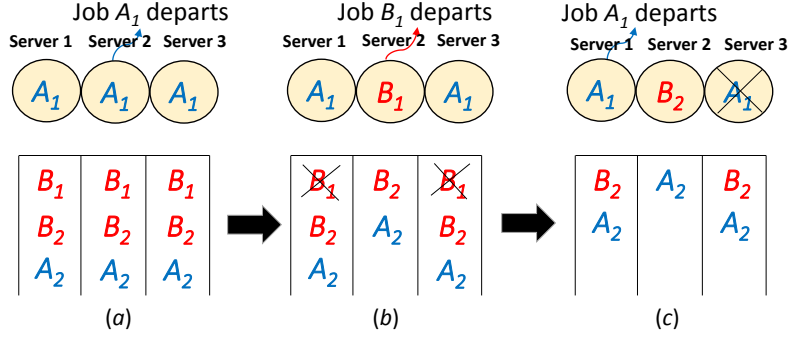


Fig. 3: System state evolution: two-class FJ system with FCFS.

Fig. 3(a)-(c) illustrates the evolution of system state under the FCFS policy. After server 2 finished job A_1 in Fig. 3(a), B_1 enters server 2 and is finished in the next state (Fig. 3(b)) while other servers still process A_1 . Since $k_B = 1$, the remaining two copies of B_1 immediately exit the system. Finally in Fig. 3(c) server 1 finishes A_1 and since $k_A = 2$, A_1 exits at server 3.

A. Latency and Energy Efficiency

Definition 1. For a $\{\lambda_i, l_i, \mu, (n, k_1, k_2, \dots, k_R)\}$ DSS, the average latency of class i under some scheduling policy \mathcal{P} is defined as

$$T_{\mathcal{P}}^i = T_{s,\mathcal{P}}^i + T_{q,\mathcal{P}}^i, \quad (1)$$

where $T_{s,\mathcal{P}}^i$ and $T_{q,\mathcal{P}}^i$ are the average service time and waiting time (in queue) for a job of class i respectively.

For a $\{\lambda_i, l_i, \mu, (n, k_1, k_2, \dots, k_R)\}$ DSS operating under scheduling policy \mathcal{P} , a relevant metric for measuring the energy efficiency, $\mathcal{E}_{\mathcal{P}}$, of the DSS is the data transfer throughput metric [13]. It is defined as the limiting ratio of the amount of data processed, $D_{\mathcal{P}}(t)$, by the DSS to the energy consumed, $E_{\mathcal{P}}(t)$, by the DSS in a infinitely large time duration t . It has units of bits/Joule. Now $D_{\mathcal{P}}(t)$ is simply,

$$D_{\mathcal{P}}(t) = \sum_{i=1}^R l_i N_i(t), \quad (2)$$

where $N_i(t)$ is the number of jobs of class i processed by DSS in a time interval t . In order to determine $E_{\mathcal{P}}(t)$, we model the power consumption of the DSS as follows:

- To reduce power consumption, the servers are equipped with the dynamic voltage/frequency scaling (DVFS) mechanism and low-power states [39]. The DVFS mechanism reduces operating voltage and CPU processing speed (or frequency) in step to reduce utilization and hence increase power savings.

- The power consumed by a server in any state is the sum of power consumed by the CPU and the platform which comprises of chipset, RAM, HDD, Fan etc. The power consumed by the CPU and platform in a given state is assumed to be same across all the n servers.
- The power consumed by a server (CPU and platform) while being in active and low-power state is denoted by P_{on} and P_{off} respectively. A server is in active mode during the busy periods (i.e., there are outstanding jobs waiting for service). In general, at the end of a busy period, a server remains active for a while and then enters a sequence of low-power states staying in each for a predetermined amount of time. For ease of analysis, we lump them into a single low-power state with constant CPU power, C_l and constant platform power, P_l . After the busy period is over, the server remains in active mode for d_l and then enters the low-power state⁴. When the busy period restarts, the server incurs a wake-up latency w_l in which it consumes active mode power, but is not capable of processing any job requests. Fig. 4 explains this using an example.
- The CPU power during active mode, C_a is proportional to $V^2 f$, where V is the supply voltage and f is the CPU operating frequency⁵ ($f \in [0, 1]$) and are set by the DVFS mechanism. Further, we assume that V is proportional to f [39]. So $C_a = C_0 f^3$, for some maximum power C_0 . The power consumed by the platform during active mode, P_a , is constant.
- $t_{\text{busy}}^{i,j,k}$ denotes the duration of time for which the k^{th} server is *busy* serving j^{th} job of i^{th} class.
- $t_{\text{idle}}^{i,j,k}$ denotes the duration of idle period after the k^{th} server finished the j^{th} job of i^{th} class.

Using the above notations, the active mode power per server is $P_{\text{on}} = C_a + P_a = C_0 f^3 + P_a$. Similarly, $P_{\text{off}} = C_l + P_l$. Consider any time duration t of interest during the operation of DSS. During this period, the total time for which the DSS is in active mode, t_a , is sum total (across all servers) of all busy periods plus the active mode time before entering low-power state. Mathematically, we have,

$$t_a = \sum_{i=1}^R \sum_{j=1}^{N_i(t)} \sum_{k=1}^n t_{\text{busy}}^{i,j,k} + \max(0, t_{\text{idle}}^{i,j,k} - d_l) \quad (3)$$

The total time for which DSS is in low-power state, t_l is,

$$t_l = nt - t_a. \quad (4)$$

We have now the following definition of energy efficiency of a DSS.

Definition 2. For a $\{\lambda_i, l_i, \mu, (n, k_1, k_2, \dots, k_R)\}$ DSS, the energy efficiency of the DSS under some scheduling policy \mathcal{P} is defined as,

$$\mathcal{E}_{\mathcal{P}} = \lim_{t \rightarrow \infty} \frac{D_{\mathcal{P}}(t)}{E_{\mathcal{P}}(t)}, \quad (5)$$

$$= \lim_{t \rightarrow \infty} \frac{\sum_{i=1}^R l_i N_i(t)}{P_{\text{on}} t_a + P_{\text{off}} t_l}, \quad (6)$$

where (6) follows from (5) using (2). The expressions for t_a and t_l are given in (3) and (4) respectively.

Next in order to highlight the relationship between the average latency and energy efficiency of a DSS, we consider the special case of a M/M/1 system and a single data-class. For tractability of analysis, here

⁴As a consequence, if the duration of idle period (time between end of a busy period and start of the next one) is smaller than d_l , then the server always remains active).

⁵Due to this, the effective service rate for class i becomes $\mu_i = f k_i \mu / l_i$.

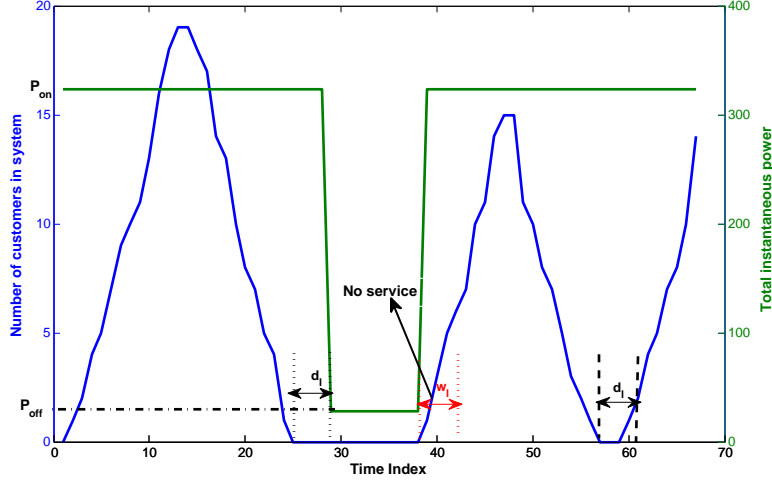


Fig. 4: Variation of total power consumption of DSS across multiple busy periods and idle periods. The switch to idle state happens only for idle periods with duration greater than d_l but it then results in a wake-up latency of w_l .

we assume that d_l , w_l and P_{off} are all 0. Then from the Definition 1 for average latency⁶, we have,

$$T = T_s + T_q, \quad (7)$$

$$= \frac{1}{\mu'} + \frac{\lambda}{\mu'(\mu' - \lambda)} = \frac{1}{\mu' - \lambda}, \quad (8)$$

where (8) follows from (7) by noting that for a M/M/1 system, the mean service time is $T_s = \frac{1}{\mu'}$ and mean waiting time is $T_q = \frac{\lambda}{\mu'(\mu' - \lambda)}$. Here, $\mu' = \frac{\mu f}{l}$ is the effective service rate. Here, The energy efficiency is computed using (6) as

$$\mathcal{E} = \lim_{t \rightarrow \infty} \frac{lN(t)}{P_{\text{on}}t_a + P_{\text{off}}t_l}, \quad (9)$$

$$= \lim_{t \rightarrow \infty} \frac{lN(t)}{P_{\text{on}} \sum_{i=1}^{N(t)} T_{s,i}}, \quad (10)$$

$$= \frac{l}{P_{\text{on}} \lim_{t \rightarrow \infty} \frac{\sum_{i=1}^{N(t)} T_{s,i}}{N(t)}}, \quad (11)$$

$$= \frac{l}{P_{\text{on}} T_s}, \quad (12)$$

where (10) follows from (9) by noting that t_{on} is sum of service time of each of $N(t)$ jobs (denoted by $T_{s,i}$ for the i^{th} job) and by neglecting the power consumed when server is idle i.e., $P_{\text{off}} = 0$. Then (12) follows from (11) from the definition of average service time. Thus the energy efficiency is inversely related to the average service time of jobs. It is difficult to find a closed form expression for the energy efficiency of a heterogeneous DSS but the general trend of inverse proportionality between latency and energy efficiency continues to hold true as verified through extensive simulations in Section VI. The average latency is also directly related to the average service time⁷. Therefore, we conclude that energy efficiency and average latency of a DSS are closely related to each other. Henceforth, we focus on the latency analysis of a heterogeneous DSS.

⁶In this special case, the scheduling policy \mathcal{P} and class index i are not relevant and hence dropped from 1.

⁷Queuing delay depends on job arrival rate and service time. So the latency which is sum of queuing delay and service time directly depends on service time.

IV. PRELIMINARIES

In this section, we first present the analysis of average service latency in a multi-class single server system with FCFS scheduling policy. For the corresponding results in a priority (preemptive/non-preemptive) queuing system, we refer the reader to [40]. To improve the tractability of the latency analysis, the analytical results in this work ignore the impact of wakeup latency w_l similar to other works in literature [17], [21]–[23], [29]. We then briefly review the existing results for upper and lower bounds on the average latency for a (n, k) homogenous Fork-Join system [17].

A. Average Latency in Multi-class Single Server System with FCFS Scheduling

Consider the system model described in Fig. 1 with $n = 1$ server and FCFS scheduling policy. The FCFS system can be modeled as a M/G/1 queuing system with net arrival rate, $\lambda = \sum_{r=1}^R \lambda_r$, and a general service distribution, S . The average latency of jobs of class i is the sum of their average waiting time (in queue) and the average service time. Let S_i be a random variable representing the service time for a job of class i in the FCFS system. Then the average service time of jobs of class i is simply the expectation, $E[S_i]$. In the FCFS system, the waiting time, W_{FCFS} , for jobs of all the classes is same and is given by the Pollaczek-Khinchine (P-K) formula [41] (for M/G/1 system) as

$$W_{\text{FCFS}} = \frac{\lambda(E[S^2])}{2(1 - \lambda E[S])}, \quad (13)$$

Therefore, the average latency for jobs of class i is,

$$T_{\text{FCFS}}^i = E[S_i] + \frac{\lambda E[S^2]}{2(1 - \lambda E[S])} = E[S_i] + \frac{\lambda(V[S] + E[S]^2)}{2(1 - \lambda E[S])}, \quad (14)$$

where $V[\cdot]$ denotes the variance of the random variable. Now the fraction of jobs of class i , p_i is

$$p_i = \frac{\lambda_i}{\sum_{r=1}^R \lambda_r} = \frac{\lambda_i}{\lambda}. \quad (15)$$

So the probability that S takes on the value of S_i is $p_i \forall i = 1, 2, \dots, R$. Therefore the probability distribution function (pdf) of S is given by

$$f_S(s) = \sum_{r=1}^R p_r f_{S_r}(s). \quad (16)$$

Then the mean and the second moment of S are simply

$$E[S] = \sum_{r=1}^R p_r E[S_r], \quad E[S^2] = \sum_{r=1}^R p_r E[S_r^2]. \quad (17)$$

Using (15) and (17) in (14), we obtain,

$$T_{\text{FCFS}}^i = E[S_i] + \frac{\sum_{r=1}^R \lambda_r [V[S_r] + E[S_r]^2]}{2 \left(1 - \sum_{r=1}^R \lambda_r E[S_r] \right)}. \quad (18)$$

B. Latency Analysis of Homogenous DSS

An exact latency analysis of the (n, k) DSS is prohibitively complex because the Markov chain has a state space with infinite states in at least k dimensions. This is exemplified in Figure 5 which shows

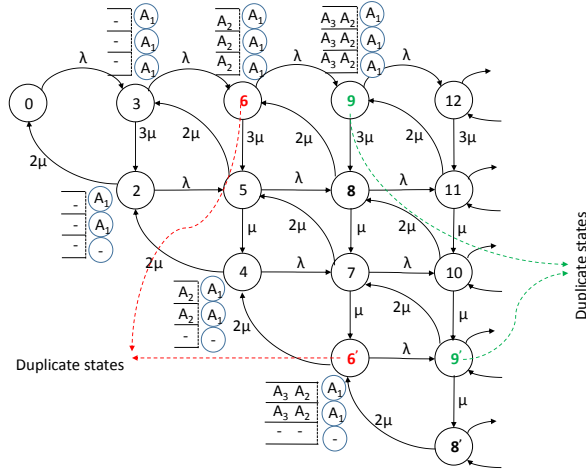


Fig. 5: Markov chain for a $(3, 2)$ Fork-Join system.

the Markov chain evolution for a $(3, 2)$ DSS. Each state is characterized by the number of jobs in the system. The arrival and service rates of jobs are λ and μ respectively. We note that as more jobs enter the system, the Markov Chain starts growing in two-dimensions and results in multiple states with the same number of jobs in the system such as states 6 and 6'. Thus, we note that an exact analysis of the F-J system is very complex. Therefore, we review existing upper- and lower-bounds for the average latency of homogenous DSS.

1) *Lower Bound on Average Latency:* In a (n, k) DSS, a job is considered finished when k out of n servers finish that job. This is equivalent to each job going through k stages sequentially, where the transition from one stage to the next occurs when one of the remaining servers finishes a sub-task of the job [42]. We note that at any stage s , the maximum possible service rate for a job that is not finished yet is $(n - s + 1)\mu'$, where $\mu' = \frac{fk\mu}{l}$. This happens when all the remaining sub-tasks of a job are at the head of their queues. Thus, we can enhance the latency performance in each stage s by approximating it with a M/M/1 system with service rate $(n - s + 1)\mu'$. Then, the average latency of the original system (denoted by T), can be lower bounded as

$$T \geq T_{LB} = \sum_{i=1}^k \frac{1}{(n - i + 1)\mu' - \lambda}, \quad (19)$$

where T_{LB} denotes the lower bound on the average latency of the F-J system.

2) *Upper Bound on Average Latency:* To upper-bound the performance of the (n, k) F-J system, we degrade its performance by approximating it with a (n, k) Split-Merge (SM) system, proposed in [17]. In the (n, k) SM system, after a server finishes a copy of a job, it is blocked and not allowed to accept new jobs until all k copies of the current job are finished. When k copies of a job are finished, the copies of that job at remaining $n - k$ servers exit the system immediately. The SM system thus can be modeled as a M/G/1 system with arrival rate λ and a service distribution that follows k^{th} order statistics [43] and is described here for reference.

Let X_1, X_2, \dots, X_n be n i.i.d random variables (rv). Now if we order the rv's in ascending order to get, $X_{1,n} < X_{2,n} \dots < X_{k,n} \dots < X_{n,n}$, then the distribution of the k^{th} smallest value, $X_{k,n}$, is called the

k^{th} order statistics. The pdf of $X_{k,n}$ is given by⁸

$$f_{X_{k,n}}(x) = \binom{n}{k-1, 1, n-k} F_X(x)^{k-1} (1 - F_X(x))^{n-k} f_X(x) \quad (20)$$

where $F_X(x)$ and $f_X(x)$ are the cumulative density function and pdf of X_i respectively for all i . The average latency of the F-J system is thus upper-bounded by the average latency for the SM system, T_{SM} as,

$$T \leq T_{\text{SM}} = \underbrace{\mathbb{E}[X_{k,n}]}_{\text{service time}} + \underbrace{\frac{\lambda [\mathbb{V}[X_{k,n}] + \mathbb{E}[X_{k,n}]^2]}{2(1 - \lambda \mathbb{E}[X_{k,n}])}}_{\text{waiting time}}, \quad (21)$$

where the average service time is simply the expectation, $\mathbb{E}[X_{k,n}]$ and the average waiting time for a M/G/1 system given by the P-K formula in (13). Now if X_i is exponential with mean $1/\mu'$ (where $\mu' = \frac{fk\mu}{l}$), then the mean and variance of $X_{k,n}$ are given by,

$$\mathbb{E}[X_{k,n}] = \frac{H_{n-k,n}^1}{\mu'}, \mathbb{V}[X_{k,n}] = \frac{H_{n-k,n}^2}{\mu'^2}, \quad (22)$$

where $H_{x,y}^z$ is a generalized harmonic number of order z defined by

$$H_{x,y}^z = \sum_{j=x+1}^y \frac{1}{j^z}, \quad (23)$$

for some positive integers x, y and z .

V. MAIN RESULTS

Section IV-B presented bounds on the average latency for the (n, k) F-J system. To extend the lower-bound result (19) to a heterogeneous FJ system, a naive approach would be to approximate it with a homogenous FJ system with jobs of class i only while evaluating the lower bound on average latency of class i . Thus a naive lower-bound on the average latency for jobs of class i is,

$$T_{\text{naive}}^i \geq \sum_{j=0}^{k_i-1} \frac{1}{(n-j)\mu_i - \lambda_i}. \quad (24)$$

This lower bound holds true irrespective of the scheduling policy used in the heterogeneous system. However, this is a loose bound as it ignores the dependency of response time for a job of class i on the jobs of other classes in the system which compete for the service at the same server.

Therefore, through a rigorous latency analysis of various scheduling policies, we next account for this inter-dependency in average latency of different classes and present lower and upper bounds for the heterogeneous FJ system. To this end, we first define a set of variables for a compact presentation of the results. The operational meaning of these variables will become clear when we present the proof of the results.

- (n, k_i) is the MDS code used to store data of class i .
- l_i is the file-size for class i .
- λ_i is the arrival rate for jobs of class i .
- $\mu_i = k_i f \mu / l_i$ is the effective service rate for jobs of class i , where μ is the service rate per unit file size.

⁸ The result in (20) can be understood as follows. First select groups of $k-1$, 1, and $n-k$ servers out of n servers in $\binom{n}{k-1, 1, n-k}$ possible ways. Then the pdf of service time for the singled-out server is simply $f_X(x)$. Now since X_i are i.i.d random variables, the probability that the selected $k-1$ servers finish their jobs before the singled-out server is $F_X(x)^{k-1}$. Similarly, the probability that $n-k$ servers finish their jobs after the singled-out server is $(1 - F_X(x))^{n-k}$.

- $\rho_i = \frac{\lambda_i}{\mu_i}$ is the server utilization factor for class i .
- $\mathcal{S}_i = \sum_{r=1}^i \rho_r H_{n-k_r, n}^1$.

A. Main Results

Lemma 1 gives the stability conditions of the heterogeneous DSS for various scheduling policies. The upper- and lower-bounds on the average latency for various scheduling policies are presented in Theorem 1 and 2 respectively.

Lemma 1. *For a $(n, k_1, k_2, \dots, k_R)$ Fork-Join system to be stable, the following condition must be satisfied at each node.*

- *FCFS scheduling*

$$\left(\sum_{r=1}^R k_r \lambda_r \right) \left(\sum_{r=1}^R \frac{\lambda_r l_r}{k_r} \right) < n f \mu \sum_{r=1}^R \lambda_r. \quad (25)$$

- *Preemptive/Non-preemptive priority scheduling*

$$\sum_{r=1}^R \lambda_r l_r < n f \mu. \quad (26)$$

Next, to upper-bound the average latency, we extend the Split-Merge (SM) system (defined in Section IV-B2) to R data classes, keeping the scheduling policy same as that for the original system. Then for a given scheduling policy, the upper-bound on average latency is basically the average latency of the corresponding SM system. This in turn is sum of the average service time and waiting time which can be determined by noting the equivalence between the SM system as a M/G/1 system as described in Section IV-B2. We thus obtain the following upper-bounds on the average latency for different scheduling policies.

Theorem 1. *The average latency for job requests of class i in a $(n, k_1, k_2, \dots, k_R)$ Fork-Join system is upper-bounded as follows:*

- *FCFS scheduling*

$$T_{\text{FCFS}}^i \leq \underbrace{\frac{H_{n-k_i, n}^1}{\mu_i}}_{\text{Service time}} + \underbrace{\frac{\sum_{r=1}^R \lambda_r [H_{n-k_r, n}^2 + (H_{n-k_r, n}^1)^2] / \mu_r^2}{2(1 - \mathcal{S}_R)}}_{\text{Waiting time}}. \quad (27)$$

The bound is valid only when $\mathcal{S}_R < 1$.

- *Non-preemptive priority scheduling⁹*

$$T_{\text{N-PQ}}^i \leq \frac{H_{n-k_i, n}^1}{\mu_i} + \frac{\sum_{r=1}^R \lambda_r [H_{n-k_r, n}^2 + (H_{n-k_r, n}^1)^2] / \mu_r^2}{2(1 - \mathcal{S}_{i-1})(1 - \mathcal{S}_i)}. \quad (28)$$

The bound is valid only when $\mathcal{S}_i < 1$.

- *Preemptive priority scheduling⁹*

$$T_{\text{PQ}}^i \leq \frac{H_{n-k_i, n}^1}{\mu_i(1 - \mathcal{S}_{i-1})} + \frac{\sum_{r=1}^i \lambda_r [H_{n-k_r, n}^2 + (H_{n-k_r, n}^1)^2] / \mu_r^2}{2(1 - \mathcal{S}_{i-1})(1 - \mathcal{S}_i)}. \quad (29)$$

⁹Without loss of generality, we set the classes in the order of decreasing priority as $1 > 2 > \dots > R$.

The bound is valid only when $\mathcal{S}_i < 1$.

We now define an additional set of variables for compact presentation of the results in Theorem 2.

- Without loss of generality, assume the classes are relabeled such that $k_1 \leq k_2 \leq \dots \leq k_R$. Then for class i , we define c_s as,

$$c_s = \begin{cases} 0, & 1 \leq s \leq k_1 \\ 1, & k_1 < s \leq k_2 \\ \vdots & \\ i-1, & k_{i-1} < s \leq k_i \end{cases}. \quad (30)$$

- At a stage s , let \mathcal{R}_s^i denote the set of classes with priority higher than class i and that have not been finished yet.
- $t_{s,i} = \frac{\lambda_i}{(n-s+1)\mu_i}$ at stage s and class i .
- $\mathcal{Z}_s^i = 1 - \sum_{r \in \mathcal{R}_s^i} t_{s,r}$ at stage s and class i .

For obtaining a lower-bound on the average latency, we enhance the performance of the original system similar to the process described in Section IV-B2. The processing of a job of class i is modeled as completing k_i sequential stages (or sub-tasks). Then we enhance the latency performance for job of class i in stage s by assuming the maximum possible service rate for it, i.e., $(n-s+1)\mu_i$. However, at stage s , there may also be unfinished sub-tasks of jobs of other classes which can be served with maximum possible service rate of $(n-s+1)\mu_j$, where $j \neq i$. Due to this, we model the performance of each enhanced stage as a M/G/1 system. We thus obtain the following lower-bounds on the average latency for different scheduling policies.

Theorem 2. The average latency for job requests of class i in a $(n, k_1, k_2, \dots, k_R)$ Fork-Join system is lower-bounded as follows:

- FCFS scheduling

$$T_{\text{FCFS}}^i \geq \sum_{s=1}^{k_i} \left(\underbrace{\frac{t_{s,i}}{\lambda_i}}_{\text{service time}} + \underbrace{\frac{\sum_{r=c_s+1}^R \frac{t_{s,r}^2}{\lambda_r}}{1 - \sum_{r=c_s+1}^R t_{s,r}}}_{\text{waiting time}} \right). \quad (31)$$

- Non-Preemptive priority scheduling⁹

$$T_{\text{N-PQ}}^i \geq \sum_{s=1}^{k_i} \left(\frac{t_{s,i}}{\lambda_i} + \frac{\sum_{r=c_s+1}^R \frac{t_{s,r}^2}{\lambda_r}}{\mathcal{Z}_s^i (\mathcal{Z}_s^i - t_{s,i})} \right). \quad (32)$$

- Preemptive priority scheduling⁹

$$T_{\text{PQ}}^i \geq \sum_{s=1}^{k_i} \left(\frac{t_{s,i}}{\lambda_i \mathcal{Z}_s^i} + \frac{\sum_{r \in \mathcal{R}_s^i \cup i} \frac{t_{s,r}^2}{\lambda_r}}{\mathcal{Z}_s^i (\mathcal{Z}_s^i - t_{s,i})} \right). \quad (33)$$

B. Proofs for FCFS scheduling

We now present the proofs for the stability condition and the bounds on average latency for the FCFS scheduling policy. The proofs for the remaining results are given in Appendix A.

1) *Proof of Lemma 1-FCFS scheduling:* Consider any server in the $(n, k_1, k_2, \dots, k_R)$ Fork-Join system. Jobs of class r enter the queue with rate λ_r . Each new job of class r exits the system when k_r sub-tasks of that job are completed. The remaining $n - k_r$ sub-tasks are then cleared from the system. Thus for each job of class r , $\frac{(n-k_r)}{n}$ fraction of the sub-tasks are deleted and hence the effective arrival rate of jobs of class r at any server is $\lambda_r (1 - \frac{n-k_r}{n}) = \frac{k_r \lambda_r}{n}$. Thus the overall arrival rate at any server, λ_{eff} , is

$$\lambda_{\text{eff}} = \sum_{r=1}^R \frac{k_r \lambda_r}{n}. \quad (34)$$

Let S denote the service distribution for a single-server FCFS system serving R data classes. Then from (17), the mean service time at a server is

$$E[S] = \sum_{r=1}^R p_r E[S_r] = \sum_{r=1}^R \frac{\lambda_r}{\mu_r \sum_{r=1}^R \lambda_r}, \quad (35)$$

where (35) follows from (15) and the assumption that the service time for a job of class i is exponential with rate μ_r . To ensure stability, the net arrival rate should be less than the average service rate at each server. Thus from (34) and (35) the stability condition of each queue is

$$\sum_{r=1}^R \frac{k_r \lambda_r}{n} < \left(\sum_{r=1}^R \frac{\lambda_r}{\mu_r \sum_{r=1}^R \lambda_r} \right)^{-1},$$

Since $\mu_r = \frac{f k_r \mu}{l_r}$ and the term $\sum_{r=1}^R \lambda_r$ is a constant, with simple algebraic manipulations we arrive at

$$\left(\sum_{r=1}^R k_r \lambda_r \right) \left(\sum_{r=1}^R \frac{\lambda_r l_r}{k_r} \right) < n f \mu \sum_{r=1}^R \lambda_r. \quad (36)$$

This completes the proof of stability condition for FCFS scheduling.

2) *Proof of Theorem 1-FCFS scheduling:* The FCFS system can be modeled as a M/G/1 queuing system with arrival rate $\lambda = \sum_{r=1}^R \lambda_r$ and a general service time distribution S . Then the average latency for a job of class i in a FCFS scheduling system is given by (18) as,

$$T_{\text{fcfs}}^i = E[S_i] + \frac{\sum_{r=1}^R \lambda_r [V[S_r] + E[S_r]^2]}{2 \left(1 - \sum_{r=1}^R \lambda_r E[S_r] \right)}.$$

To obtain an upper bound on the average latency, we degrade the FJ system in the following manner. For a job of class i , the servers that have finished processing a sub-task of that job are blocked and do not accept new jobs until k_i sub-tasks of that job have been completed. Then the sub-tasks at remaining $n - k_i$ servers exit the system immediately. Fig. 6 illustrates this process using Example 1. When A_1 is finished at server 2, it is blocked (see Fig. 7(b)) until another $k_A = 2$ copies are finished. Now this performance-degraded system can be modeled as a M/G/1 system where the distribution of the service process, S_i , follows k_i^{th} ordered statistics as described in Section IV-B2. Now for any class i , the service

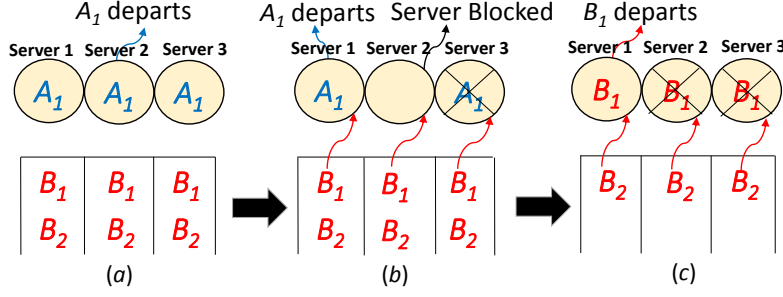


Fig. 6: Enhanced two-class FJ system with FCFS.

time at each of the n servers is exponential with mean $1/\mu_i$. Hence from (22), the mean and variance of S_i are,

$$\mathbb{E}[S_i] = \frac{H_{n-k_i,n}^1}{\mu_i}, \quad \mathbb{V}[S_i] = \frac{H_{n-k_i,n}^2}{\mu_i^2}. \quad (37)$$

Substituting (37) in (18), we get the following upper bound on average latency:

$$T_{\text{FCFS}}^i \leq \underbrace{\frac{H_{n-k_i,n}^1}{\mu_i}}_{\text{service time}} + \underbrace{\frac{\sum_{r=1}^R \lambda_r [H_{n-k_r,n}^2 + (H_{n-k_r,n}^1)^2] / \mu_r^2}{2(1 - \mathcal{S}_R)}}_{\text{waiting time}}, \quad (38)$$

where $\mathcal{S}_R = \sum_{r=1}^R \rho_r H_{n-k_r,n}^1$ and $\rho_r = \lambda_r / \mu_r$. This concludes the proof of upper bound on the average latency for FCFS scheduling.

3) *Proof of Theorem 2-FCFS scheduling:* For the purpose of obtaining a lower bound on the average latency of class i , using insights from Section IV-B1, we map the parallel processing in the proposed FJ system to a sequential process consisting of k_i processing stages for k_i sub-tasks of a job of class i . The transition from one stage to the next occurs when one of the remaining servers finishes a sub-task of the job. Let c_s denotes the number of classes that are finished before start of stage s , defined in (30). The processing in each stage s corresponds to a single-server FCFS system with jobs of all but classes $1, 2, \dots, c_s$. Then, using (14) for the FCFS sub-system at stage s , the average latency for a sub-task of a job of class i in stage s is given by,

$$T_{\text{FCFS},s}^i = \mathbb{E}[S_i^s] + \frac{\lambda \mathbb{E}[(S^s)^2]}{2(1 - \lambda \mathbb{E}[S^s])}, \quad (39)$$

where S^s is a r.v. denoting the service time for any sub-task in stage s and S_i^s denotes the service time for a sub-task of class i in stage s . Now the moments of S^s and S_i^s are related to each other in the same way as the moments of S and S_i in (17). So we have,

$$\mathbb{E}[S^s] = \sum_{r=c_s+1}^R p_r \mathbb{E}[S_r^s], \quad \mathbb{E}[(S^s)^2] = \sum_{r=c_s+1}^R p_r \mathbb{E}[(S_r^s)^2]. \quad (40)$$

Substituting (40) in (39), we get

$$T_{\text{FCFS},s,c_s}^i = \mathbb{E}[S_i^s] + \frac{\sum_{r=c_s+1}^R \lambda_r \mathbb{E}[(S_i^s)^2]}{2 \left(1 - \sum_{r=c_s+1}^R \lambda_r \mathbb{E}[S_i^s] \right)}. \quad (41)$$

Now we note that at any stage s , the maximum possible service rate for a job of class j that is not finished yet is $(n - s + 1)\mu_j$. This happens when all the remaining sub-tasks of job of class j are at the head of their buffers. Thus, we can enhance the latency performance in each stage s by approximating it with a M/G/1 system with service rate $(n - s + 1)\mu_j$ for jobs of class j . Then, the average latency for sub-task of job of class i in stage s is lower bounded as,

$$T_{\text{FCFS},s,c_s}^i \geq \frac{1}{(n - s + 1)\mu_i} + \frac{\sum_{r=c_s+1}^R \frac{\lambda_r}{(n-s+1)\mu_r^2}}{1 - \sum_{r=c_s+1}^R \frac{\lambda_r}{(n-s+1)\mu_r}}, \quad (42)$$

Finally, the average latency for class i in this enhanced system is simply $\sum_{s=1}^{k_i} T_{\text{FCFS},s,c_s}^i$. This gives us

$$T_{\text{FCFS}}^i \geq \underbrace{\sum_{s=1}^{k_i} \frac{t_{s,i}}{\lambda_i}}_{\text{service time}} + \underbrace{\sum_{s=1}^{k_i} \left(\frac{\sum_{r=c_s+1}^R \frac{t_{s,r}}{(n-s+1)\mu_r}}{1 - \sum_{r=c_s+1}^R \frac{t_{s,r}}{(n-s+1)\mu_r}} \right)}_{\text{waiting time}},$$

where $t_{s,i} = \frac{\lambda_i}{(n-s+1)\mu_i}$. This concludes the proof of lower bound on the average latency for FCFS scheduling.

VI. QUANTITATIVE RESULTS AND DISCUSSION

In this section, we use Monte-Carlo simulations of a heterogeneous Fork-Join system to study the impact of varying various system parameters on the average latency of different classes and the energy efficiency of DSS. For simplicity, the number of data classes is set to 2. Data of class 1 is stored using $(n, k_1) = (10, 5)$ MDS code. Data of class 2 is stored using $(10, k_2)$ MDS code where k_2 is varied from 1 to 10. Arrival rates for the two classes are set as: $\lambda_1 = 0.15$ and $\lambda_2 = 0.5$. The job size for both the classes is set to 1 kilobits. Job requests for both the classes are served using full redundancy (i.e., $r_1 = r_2 = n$). We set the power consumption parameters by using the data for Intel Xeon family of CPUs¹⁰ and associated platform components [39]. Therefore, we set $C_0 = 203.13$ W, $P_a = 120$ W, $C_l = 15$ W, $w_l = 6$ s, and $P_l = 13.1$ W. The CPU frequency, f is set to 1 unless mentioned otherwise.

A. Impact of fault-tolerance and service rate

The behavior of average latency with respect to change in fault-tolerance k , is governed by two opposing factors.

- Increasing k reduces the number of servers available for serving the next job in queue, thus resulting in an increase in latency.
- Increasing k increases the effective service rate ($k\mu$) of each server as each server stores a smaller fraction ($\frac{1}{k}$) of the job. This decreases the average latency.

Fig. 7 shows the average latency for jobs of class 2 versus k_2 for the FCFS system with $\mu = 1/6$ and 1^{11} . The file size for both classes are equal to 1 kb. We note that the average latency increases on increasing k_2 . This is because μ is large enough, so the increment in latency due to the first factor dominates the decrease in latency due to the second factor. We also note that the bounds are somewhat loose at high values of k_2 and low values of μ . In particular, the lower bound becomes loose because at each processing

¹⁰We use the power consumption parameters of “Deeper Sleep” state for our low-power state.

¹¹In our work, we set file size to be multiples of 1 kilobits. So μ is defined per kilobit.

stage of the serial Fork-Join system, the difference between the actual service rate and its bound at s^{th} stage of processing (i.e. $(n - s + 1)\mu_i$) for jobs of class i increases with increase in k and decrease in μ . Similarly the upper bound becomes worse because the service time lost due to blocking increases significantly at low μ and high k values. This is because the remaining sub-tasks are served really slow (low μ) and the blocking continues until a large number of sub-tasks (high k) are finished. Finally, as expected, we note that the naive lower bound on latency of class 2 is loose as compared to the proposed lower bound for the FCFS system.

B. Impact of coding on energy efficiency and storage space

Fig. 8 illustrates the impact of varying the code rate (k/n for (n, k) MDS code) on the latency, energy efficiency and network bandwidth of system. At one extreme is the $(n, 1)$ replication code with code rate $1/n$ that has minimum latency (see Fig. 7) and maximum energy efficiency. This is because we just wait for any one server to finish the job. For a fixed n , low latency translates to higher data throughput and hence higher energy efficiency. However, the total storage space per file for $(n, 1)$ code is nl where l is file size. Hence the (write) network bandwidth is maximum at $k = 1$. At the other extreme is (n, n) code with no fault-tolerance and no storage overhead (storage size is l). But it suffers from high latency and low energy efficiency. This is because we need to wait for all the servers to finish their sub-tasks of a job before the job is completed. Hence latency and throughput suffers which in turn decreases energy efficiency.

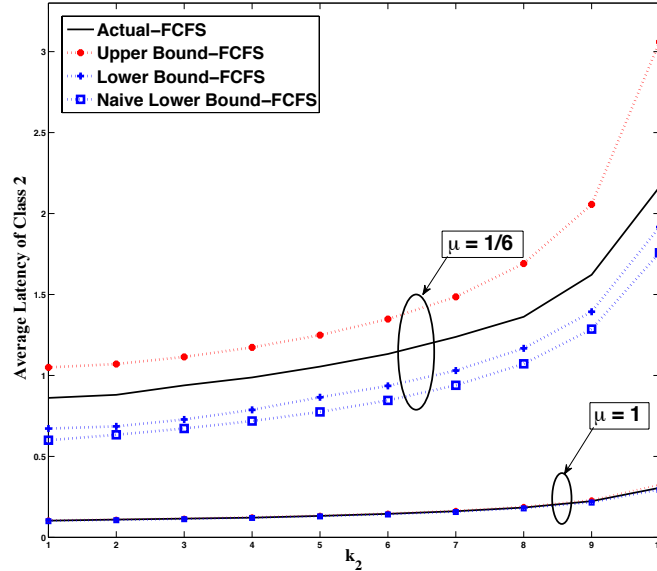


Fig. 7: Latency of a data-class increases with increase in its code-rate and decreases with increase in service rate.

C. Impact of number of servers in DSS

Fig. 9 shows the impact of increasing the number of servers (n) on the latency and energy efficiency of DSS, while all other system parameters are kept constant. We observed that for low values of n , increasing n increases the energy efficiency. This is because of more servers available to serve the job which reduces average latency and thus increase the throughput. The increase in throughput due to lower latency outweighs the increase in energy consumption due to higher n . Hence the overall effect is that energy efficiency increases. However at high values of n , increasing n results in diminishing returns in latency and throughput. This is because latency improvement is limited by effective service rate ($k\mu/l$) and not the number of servers. At very large n , the energy consumption becomes quite significant. Therefore, the energy efficiency begins to decrease at large n . We thus conclude that **there is an optimum value of n that maximizes energy efficiency and has near minimal latency.**

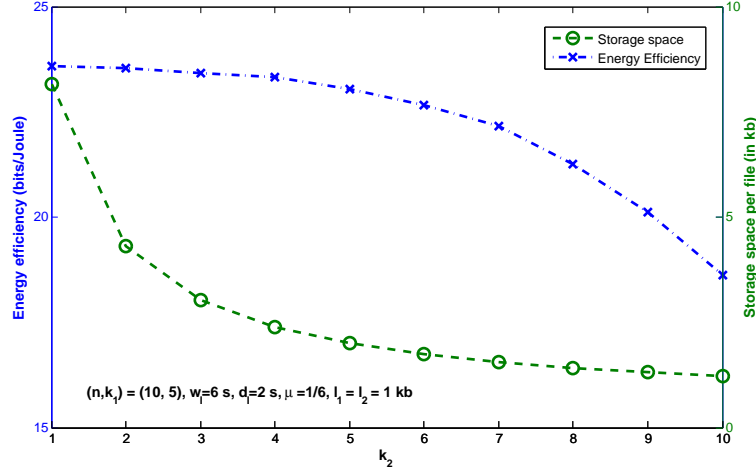


Fig. 8: Tradeoff between energy efficiency of DSS and storage space per file with variation in code-rate.

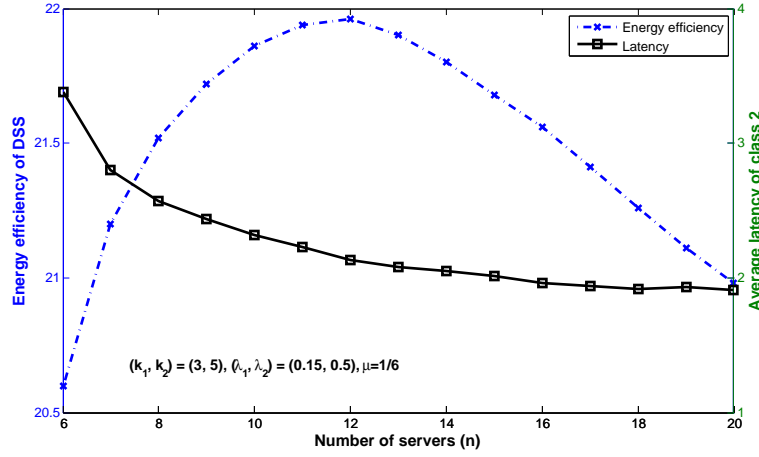


Fig. 9: Energy efficiency increases and attains a maxima as number of servers is increased while latency behaves in an inverse fashion.

D. Impact of general service time

In most of the practical DSS, the service times are not exponentially distributed but rather have heavy-tail which means that there is a significant probability of very large service times. Pareto distribution has been found to be a good fit for service time distribution in practical DSS [44], [45]. Its cumulative distribution function is given by

$$F_S(s) = \begin{cases} 0 & \text{for } s < s_m \\ 1 - \left(\frac{s_m}{s}\right)^\alpha & \text{for } s \geq s_m \end{cases} \quad (43)$$

Here α is shape parameter and s_m is the scale parameter. As the value of α decreases the service becomes more heavy-tailed and it becomes infinite for $\alpha \leq 1$. Figures 10 and 11 show the impact of Pareto service distribution on the latency and energy efficiency of DSS for $\alpha = 1.1$ and 6 respectively. At $\alpha = 6$, the service distribution is not very heavy-tailed. So increasing k_2 reduces latency of jobs of class 2 due to increase in their effective service rate ($k_2 \mu f / l_2$). However, at $\alpha = 1.1$, the service time distribution becomes very heavy-tailed, so as k_2 becomes large, the increase in service time due to waiting for more servers (larger k) outweighs the decrease due to higher effective service rate. In both cases, we note that latency behaves inversely to the change in latency. We note that as k_2 increases from 1 to 10, energy efficiency first starts increasing, reaches a maximum and then starts decreasing for large k . We conclude that **for heavy-tailed service distribution, there exists an optimal code-rate that yield maximum**

energy efficiency and minimum latency for heavy-tailed service times.

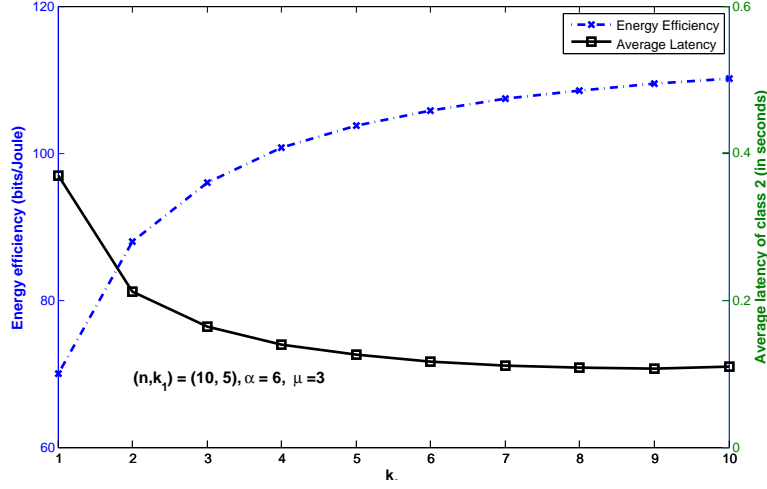


Fig. 10: A light tailed general service distribution (Pareto-distribution with $\alpha = 6$) results in monotonically decreasing latency as a function of code-rate. Energy efficiency follows an inverse behavior.

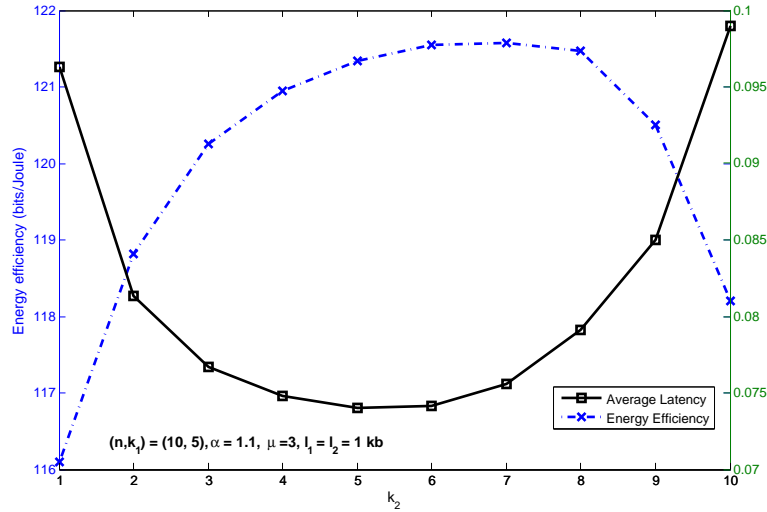


Fig. 11: A heavy tailed general service distribution (Pareto-distribution with $\alpha = 1.1$) results in minimal latency and maximum energy efficiency point as the code-rate is increased.

E. Impact of heavy-tailed arrival distribution

Fig. 12 illustrates the impact of a general (Pareto) arrival time distribution on the latency and energy efficiency of DSS. We observed that when distribution becomes heavy tailed, latency increases (and energy efficiency decreases) with increase in code rate. The heavy-tailed arrival distribution results in occasional very large inter-arrival time, however the arrival rate remains the same. Since it does not influence significantly the service dynamics, we observe that the latency increases with increase in code-rate similar to the M/M/1 case (in Fig. 7). Since latency increases, energy efficiency decreases with increase code-rate similar to previous results.

F. Impact of number of redundant requests

We now explore the impact of varying the number of redundant requests (i.e., sending job requests to more than k servers) on the average latency and energy efficiency of DSS. The behavior of latency is governed by two opposing factors.

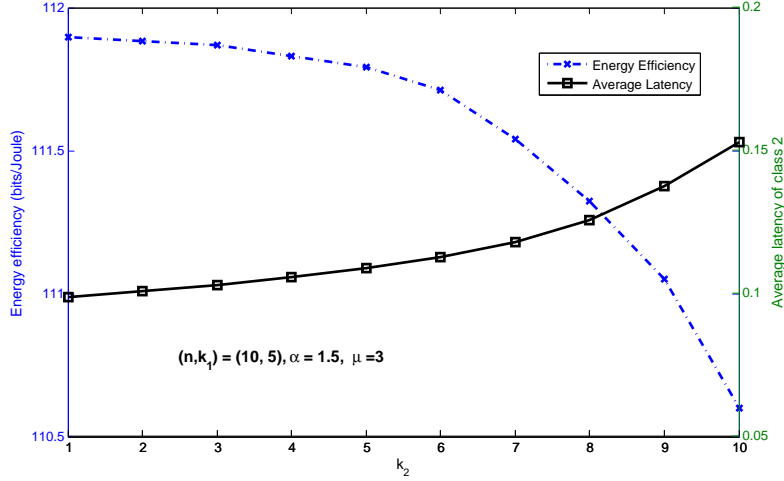


Fig. 12: A heavy tailed inter-arrival time distribution (Pareto-distribution with $\alpha = 1.5$) results in monotonically increasing latency (and monotonically decreasing energy efficiency) as the code-rate is increased.

- Increasing the number of redundant requests reduces the service time because there are more servers available that simultaneously process the same job. This reduces the service time of each job. It increases the energy efficiency because the servers can process more requests per unit time.
- On the other hand, increasing the number of redundant requests reduces the number of servers available for serving the next job in queue, thus resulting in increase of size of queue at the servers. This results in loss of throughput and hence a plausible decrease in energy efficiency.

As it turns out that the first factor is more dominant than the second one, thereby resulting in an overall reduction in latency (increase in energy efficiency) by increasing the number of redundant requests. This behavior can be observed in Fig. 13 which shows the average latency of class 1 and energy efficiency of DSS for FCFS scheduling. In this figure, the redundancy for class 1, r_1 , is varied from 4 to 10 and the redundancy of class 2 is set to $r_2 = 10$.

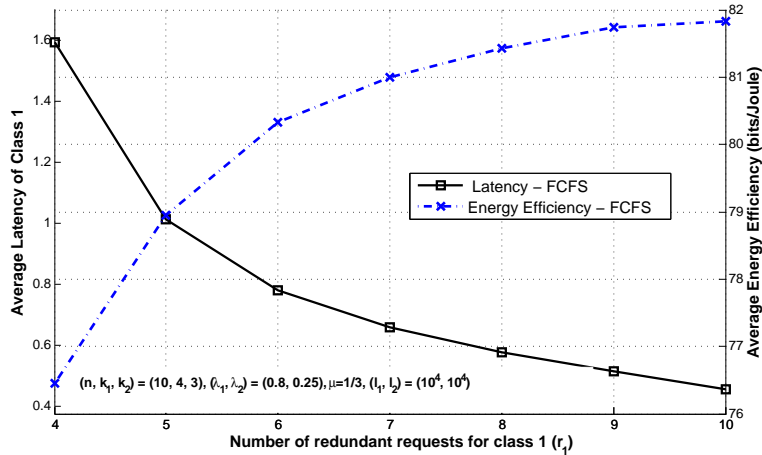


Fig. 13: Sending redundant requests reduces average latency and improves energy efficiency.

VII. CONCLUSIONS

In this paper, we proposed a novel multi-tenant DSS model and analyzed the energy efficiency of the system via lens of system latency. In the proposed heterogeneous DSS, each data class can possibly have different job arrival rate, job size and its data can be stored with a different fault-tolerance requirement by coding it with appropriate (n, k) MDS code. In order to evaluate the impact of various parameters

of DSS on its energy efficiency, we defined a *data throughput* based energy efficiency metric for any given scheduling policy. We analytically established that the energy efficiency of DSS is inversely related to the system latency for a special case. This motivated us to further investigate the impact of various parameters on the relationship between the latency and energy efficiency of the DSS. Therefore, using a queuing-theoretic approach, we obtained bounds on the average latency for FCFS, preemptive and non-preemptive priority queuing policies. We verified the accuracy of the bounds for different settings of system parameters. The bounds, in general, are tight at high values of service rate, μ and low values of k . We also noted that the proposed lower bounds are tighter than a naive lower bound that follows directly from the work in [17].

Using simulations, we investigate the relationship between average latency of data classes and energy efficiency of DSS under various setting of system parameters. We found that increasing the coding rate reduces the network bandwidth but increases latency and decreases energy efficiency. We also found that there exists an optimal number of servers which maximizes energy efficiency and results in near minimal latency. We observed that for heavy-tailed service distribution (which is the case for practical DSS), there exists an optimal code-rate that yield maximum energy efficiency and minimum latency. Lastly, we studied the impact of sending redundant requests on the average latency of that data class and the energy efficiency of DSS. We noted that increasing redundancy for a data class helps to reduce its average latency and as a consequence, the overall latency decreases and energy efficiency of DSS increases.

APPENDIX A

STABILITY CONDITION AND BOUNDS ON AVERAGE LATENCY

A. Proof of Lemma 1: Stability Condition-Priority queuing scheme

Consider any node in the $(n, k_1, k_2, \dots, k_R)$ Fork-Join system. Jobs of class r enter the queue with rate λ_r . Each new job of class r exits the system when k_r sub-tasks of that job are completed. The remaining $n - k_r$ sub-tasks are then cleared from the system. Thus for each job of class r , $\frac{(n-k_r)}{n}$ fraction of the sub-tasks are deleted and hence the effective arrival rate of jobs of class r at any node is $\lambda_{\text{eff}}^r = \lambda_r \left(1 - \frac{n-k_r}{n}\right) = \frac{k_r \lambda_r}{n}$. The stability condition for a priority queue is that the overall server utilization should be less than 1. If the condition is violated, the queues belonging to a priority level lower than some limit k will grow without bound [40, Section 3.5.3]. Mathematically the stability condition is,

$$\sum_{r=1}^R \rho_r < 1, \quad (44)$$

where $\rho_r = \frac{\lambda_{\text{eff}}^r}{\mu_r}$ is the server utilization factor for class r . Substituting the expression for ρ_r with $\mu_r = \frac{fk_r\mu}{l_r}$ in (44) and rearranging terms, we get the stability condition of each queue as

$$\sum_{r=1}^R \lambda_r l_r < nf\mu. \quad (45)$$

B. Proof of Theorem 1

1) *Upper Bound - Non-preemptive priority scheme:* Let S_i be a random variable representing the service time for a job of class i . Then the average latency for a job of class i in a non-preemptive priority

scheduling system¹² is given by [40, Eq.(3.83)],

$$T_{\text{N-PQ}}^i = E[S_i] + \frac{\sum_{r=1}^R \lambda_r E[S_r^2]}{2(1 - \sum_{r=1}^{i-1} \lambda_r E[S_r])(1 - \sum_{r=1}^i \lambda_r E[S_r])}, \quad (46)$$

$$= E[S_i] + \frac{\sum_{r=1}^R \lambda_r ((E[S_i])^2 + V[S_i])}{2(1 - \sum_{r=1}^{i-1} \lambda_r E[S_r])(1 - \sum_{r=1}^i \lambda_r E[S_r])}. \quad (47)$$

To obtain an upper bound on the average latency, we degrade the FJ system in the following manner. For a job of class i , the servers that have finished processing a sub-task of that job are blocked and do not accept new jobs until k_i sub-tasks of that job have been completed. Then the sub-tasks at remaining $n - k_i$ servers exit the system immediately. For jobs of class i , this performance-degraded system can be modeled as a M/G/1 system where the distribution of the service process, S_i , follows k_i^{th} ordered statistics as described in Section IV-B2. Now for any class i , the service time at each of the n servers is exponential with mean $1/\mu_i$. So the mean and variance of S_i are given by (37). Substituting (37) in (47), we get the following upper bound on average latency:

$$T_{\text{N-PQ}}^i \leq \underbrace{\frac{H_{n-k_i,n}^1}{\mu_i}}_{\text{service time}} + \underbrace{\frac{\sum_{r=1}^R \lambda_r [H_{n-k_r,n}^2 + (H_{n-k_r,n}^1)^2]/\mu_r^2}{2(1 - \mathcal{S}_{i-1})(1 - \mathcal{S}_i)}}_{\text{waiting time}}, \quad (48)$$

Average latency of degraded system

where $\mathcal{S}_i = \sum_{r=1}^i \rho_r H_{n-k_r,n}^1$ and $\rho_r = \lambda_r/\mu_r$.

2) *Upper Bound-Preemptive priority scheme:* Let S_i be a random variable representing the service time for a job of class i . Then the average latency for a job of class (priority level) i in a preemptive priority scheduling system¹² is given by [40, Eq.(3.87)],

$$T_{\text{PQ}}^i = \frac{E[S_i]}{1 - \sum_{r=1}^{i-1} \lambda_r E[S_r]} + \frac{\sum_{r=1}^i \lambda_r E[S_r^2]}{2 \left(1 - \sum_{r=1}^{i-1} \lambda_r E[S_r]\right) \left(1 - \sum_{r=1}^i \lambda_r E[S_r]\right)}, \quad (49)$$

$$= \frac{E[S_i]}{1 - \sum_{r=1}^{i-1} \lambda_r E[S_r]} + \frac{\sum_{r=1}^i \lambda_r ((E[S_i])^2 + V[S_i])}{2 \left(1 - \sum_{r=1}^{i-1} \lambda_r E[S_r]\right) \left(1 - \sum_{r=1}^i \lambda_r E[S_r]\right)}. \quad (50)$$

To obtain an upper bound on the average latency, we degrade the FJ system in the following manner. For a job of class i , the servers that have finished processing a sub-task of that job are blocked and do not accept new jobs until k_i sub-tasks of that job have been completed. Then the sub-tasks at remaining $n - k_i$ servers exit the system immediately. For jobs of class i , this performance-degraded system can be modeled as a M/G/1 system where the distribution of the service process, S_i , follows k_i^{th} ordered statistics as described in Section IV-B2. Now for any class i , the service time at each of the n servers is exponential with mean $1/\mu_i$. Hence the mean and variance of S_i are given by (37). Substituting (37) in (50), we get

¹²The decreasing order of class priority is $1 > 2 > \dots > R$.

Class Id (i)	MDS code	Class Id (i)	Subtasks left	R_1^i
1	(10, 8)	5	4	{1,2,3,4}
2	(10, 5)	2	5	{1}
3	(10, 6)	3	6	{1,2}
4	(10, 7)	4	7	{1,2,3}
5	(10, 4)	1	8	{ ϕ }

(a) (b)

Class Id (i)	Subtasks left	R_5^i	Class Id (i)	Subtasks left	R_6^i
2	1	{1}	3	1	{1}
3	2	{1,2}	4	2	{1,3}
4	3	{1,2,3}	1	3	{ ϕ }
1	4	{ ϕ }			

(c) (d)

Fig. 14: Example illustrating the operational meaning of variable \mathcal{R}_s^i , where i is the class id and s is the processing stage. The decreasing order of class priority is $1 > 2 > 3 > 4 > 5$.

the following upper bound on average latency:

$$T_{\text{PQ}}^i \leq \underbrace{\frac{H_{n-k_i,n}^1}{\mu_i(1-\mathcal{S}_{i-1})}}_{\text{service time}} + \underbrace{\frac{\sum_{r=1}^i \lambda_r [H_{n-k_r,n}^2 + (H_{n-k_r,n}^1)^2] / \mu_r^2}{2(1-\mathcal{S}_{i-1})(1-\mathcal{S}_i)}}_{\text{waiting time}}. \quad (51)$$

where $\mathcal{S}_i = \sum_{r=1}^i \rho_r H_{n-k_r,n}^1$ and $\rho_r = \lambda_r / \mu_r$.

C. Proof of Theorem 2

1) *Lower Bound - Non-preemptive priority scheme:* For the purpose of obtaining a lower bound on the average latency of class i , using insights from Section IV-B1, we map the parallel processing in the heterogeneous FJ system to a sequential process consisting of k_i processing stages for k_i sub-tasks of a job of class i . The transition from one stage to the next occurs when one of the remaining servers finishes a sub-task of the job. Since classes are relabeled such that $k_1 \leq k_2 \leq \dots \leq k_R$, all jobs of classes 1 to i get finished when stage k_i is finished. However due to this relabeling of classes, this new class 1 is not necessarily the one with highest priority.

Let c_s denote the number of classes that are finished before start of stage s , given by (30). At any stage s , let \mathcal{R}_s^i denote the set of classes with priority higher than class i and have atleast one sub-task remaining to be completed. Fig. 14 illustrates the operational meaning of \mathcal{R}_s^i using a toy example. Fig. 13(a) specifies the MDS codes corresponding to 5 data classes and suppose we are interested in the latency of class 3 marked in red. Fig. 13(b) shows the state of the system in stage $s = 1$. The classes are reordered in order of increasing k values. Since $s = 1$, no class is finished yet and $c_1 = 0$. The last column shows the set \mathcal{R}_1^i for a class i with unfinished jobs. Fig. 13(c) shows the state of system at stage $s = 5$. Since $k_5 = 4$, all jobs of class 5 are finished and exits the system, so $c_5 = 1$. However since class 5 has lowest priority, \mathcal{R}_5^i for a remaining class i is same as \mathcal{R}_1^i . Fig. 13(d) shows the state of the system at stage, $s = 6$. Since $k_2 = 5$, all jobs of class 2 are finished and exits the system, so $c_6 = 2$. Since class 2 is higher priority than class 3, \mathcal{R}_6^3 is now $\{1\}$.

Now using (46), we get the mean completion time for sub-task of a job of class i in stage s as,

$$T_{\text{N-PQ},s,c_s,\mathcal{R}_s^i}^i = \mathbb{E}[S_i^s] + \frac{\sum_{r=c_s+1}^R \lambda_r \mathbb{E}[(S_r^s)^2]}{2 \left(1 - \sum_{r \in \mathcal{R}_s^i} \lambda_r \mathbb{E}[S_r^s]\right) \left(1 - \lambda_i \mathbb{E}[S_i^s] - \sum_{r \in \mathcal{R}_s^i} \lambda_r \mathbb{E}[S_r^s]\right)}, \quad (52)$$

where S_i^s is a random variable denoting the service time for a sub-task of class i in stage s . Unlike (46), the summation in the numerator starts from $c_s + 1$ because classes 1 through c_s have been completed at stage s . Also unlike (46), the summation in the denominator is over the set $\mathcal{R}_s^i \subseteq \{1, 2, \dots, i-1\}$ because some of the higher priority (relative to class i) classes may have been finished at start of stage s .

Now at any stage s , the maximum possible service rate for a job of class j that is not finished yet is $(n-s+1)\mu_j$. This happens when all the remaining sub-tasks of job of class j are at the head of their buffers. Thus, we can enhance the latency performance in each stage s by approximating it with a M/M/1 system with service rate $(n-s+1)\mu_j$ for jobs of class j . The average latency for sub-task of job of class i in stage s is thus lower bounded as,

$$T_{\text{N-PQ},s,c_s,\mathcal{R}_s^i}^i \geq \underbrace{\frac{1}{(n-s+1)\mu_i}}_{\text{service time}} + \underbrace{\frac{\sum_{r=c_s+1}^R \frac{\lambda_r}{(n-s+1)\mu_r^2}}{\left(1 - \sum_{r \in \mathcal{R}_s^i} \frac{\lambda_r}{(n-s+1)\mu_r}\right) \left(1 - \frac{\lambda_i}{(n-s+1)\mu_i} - \sum_{r \in \mathcal{R}_s^i} \frac{\lambda_r}{(n-s+1)\mu_r}\right)}}_{\text{waiting time}}.$$

Finally, the average latency for class i in this enhanced system is simply $\sum_{s=1}^{k_i} T_{\text{N-PQ},s,c_s,\mathcal{R}_s^i}^i$. Thus we have,

$$T_{\text{N-PQ}}^i \geq \sum_{s=1}^{k_i} \left(\frac{t_{s,i}}{\lambda_i} + \frac{\sum_{r=c_s+1}^R \frac{t_{s,r}}{(n-s+1)\mu_r}}{\mathcal{Z}_s^i (\mathcal{Z}_s^i - t_{s,i})} \right), \quad (53)$$

where $t_{s,i} = \frac{\lambda_i}{(n-s+1)\mu_i}$ and $\mathcal{Z}_s^i = 1 - \sum_{r \in \mathcal{R}_s^i} t_{s,r}$.

2) *Lower Bound - Preemptive priority scheme:* For the purpose of obtaining a lower bound on the average latency of class i , using insights from [42], we map the parallel processing in the heterogeneous FJ system to a sequential process consisting of k_i processing stages for k_i sub-tasks of a job of class i . The transition from one stage to the next occurs when one of the remaining servers finishes a sub-task of the job. Since classes are relabeled such that $k_1 \leq k_2 \leq \dots \leq k_R$, all jobs of classes 1 to i get finished when stage k_i is finished. However due to this relabeling of classes, this new class 1 is not necessarily the one with highest priority.

Let c_s denote the number of classes that are finished before start of stage s , given by (30). At any stage s , let \mathcal{R}_s^i denote the set of classes with priority higher than class i and have atleast one sub-task remaining to be completed. The operational meaning of \mathcal{R}_s^i is explained in Fig. 14. Then using (49), the

mean completion time for sub-task of a job of class i in stage s is given by,

$$T_{PQ,s,c_s,\mathcal{R}_s^i}^i = \frac{E[S_i^s]}{1 - \sum_{r \in \mathcal{R}_s^i} \lambda_r E[S_r^s]} + \frac{\lambda_i E[(S_i^s)^2] + \sum_{r \in \mathcal{R}_s^i} \lambda_r E[(S_r^s)^2]}{2 \left(1 - \sum_{r \in \mathcal{R}_s^i} \lambda_r E[S_r^s]\right) \left(1 - \lambda_i E[S_i^s] - \sum_{r \in \mathcal{R}_s^i} \lambda_r E[S_r^s]\right)}, \quad (54)$$

where S_i^s is a random variable denoting the service time for a sub-task of class i in stage s . Unlike (49), the summation terms in the numerator and denominator are over the set $\mathcal{R}_s^i \subseteq \{1, 2, \dots, i-1\}$ because some of the higher priority (relative to class i) classes may have been finished at start of stage s .

Now we note that at any stage s , the maximum possible service rate for a job of class j that is not finished yet is $(n-s+1)\mu_j$. This happens when all the remaining sub-tasks of job of class j are at the head of their buffers. Thus, we can enhance the latency performance in each stage s by approximating it with a M/M/1 system with service rate $(n-s+1)\mu_j$ for jobs of class j . The average latency for sub-task of job of class i in stage s is thus lower bounded as,

$$T_{PQ,s,c_s,\mathcal{R}_s^i}^i \geq \underbrace{\frac{1}{(n-s+1)\mu_i \left(1 - \sum_{r \in \mathcal{R}_s^i} \frac{\lambda_r}{(n-s+1)\mu_r}\right)}}_{\text{service time}} + \underbrace{\frac{\frac{\lambda_i}{(n-s+1)\mu_i^2} + \sum_{r \in \mathcal{R}_s^i} \frac{\lambda_r}{(n-s+1)\mu_r^2}}{\left(1 - \sum_{r \in \mathcal{R}_s^i} \frac{\lambda_r}{(n-s+1)\mu_r}\right) \left(1 - \frac{\lambda_i}{(n-s+1)\mu_i} - \sum_{r \in \mathcal{R}_s^i} \frac{\lambda_r}{(n-s+1)\mu_r}\right)}}_{\text{waiting time}}.$$

Finally, the average latency for class i in this enhanced system is simply $\sum_{s=1}^{k_i} T_{PQ,s,c_s,\mathcal{R}_s^i}^i$. Thus we have,

$$T_{PQ}^i \geq \sum_{s=1}^{k_i} \left(\frac{t_{s,i}}{\lambda_i \mathcal{Z}_s^i} + \frac{1 - \mathcal{Z}_s^i + \frac{t_{s,i}}{(n-s+1)\mu_i}}{\mathcal{Z}_s^i (\mathcal{Z}_s^i - t_{s,i})} \right), \quad (55)$$

where $t_{s,i} = \frac{\lambda_i}{(n-s+1)\mu_i}$ and $\mathcal{Z}_s^i = 1 - \sum_{r \in \mathcal{R}_s^i} t_{s,r}$.

REFERENCES

- [1] A. Kumar, R. Tandon, and T. Clancy, "On the latency of heterogeneous mds queue," in *IEEE Global Communications Conference (GLOBECOM)*, Dec 2014, pp. 2375–2380.
- [2] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *International Workshop on Peer-to-Peer Systems*, 2002, pp. 328–338.
- [3] "Colossus, successor to google file system," July 2010. [Online]. Available: <http://goo.gl/cUXcSm>
- [4] "Saving capacity with hdfs raid," June 2014. [Online]. Available: <http://goo.gl/P5usvs>
- [5] C. Huang, H. Simitci, Y. Xu *et al.*, "Erasure coding in windows azure storage," in *USENIX Conference on Annual Technical Conference*, 2012, pp. 2–2.
- [6] "Cisco visual networking index: Global mobile data traffic forecast update, 20132018," Feb 2014. [Online]. Available: <http://goo.gl/ULXROo>
- [7] Y. Sverdlik, "Global data center energy use to grow by 19% in 2012," Sep. 2011. [Online]. Available: <http://goo.gl/Ck1TxB>
- [8] D. Harnik, D. Naor, and I. Segall, "Low power mode in cloud storage systems," in *IEEE International Symposium on Parallel Distributed Processing*, May 2009, pp. 1–8.
- [9] D. Colarelli and D. Grunwald, "Massive arrays of idle disks for storage archives," in *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, 2002, pp. 1–11.
- [10] A. Verma, R. Koller, L. Useche, and R. Rangaswami, "Srcmap: Energy proportional storage using dynamic consolidation," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, 2010, pp. 20–20.
- [11] H. Jo, Y. Kwon, H. Kim, E. Seo, J. Lee, and S. Maeng, "Ssd-hdd-hybrid virtual disk in consolidated environments," in *Proceedings of the 2009 International Conference on Parallel Processing*, 2010, pp. 375–384.
- [12] D. Chen, E. Henis, R. I. Kat *et al.*, "Usage centric green performance indicators," *SIGMETRICS Perform. Eval. Rev.*, vol. 39, no. 3, pp. 92–96, Dec. 2011.
- [13] G. Schulz, *The Green and Virtual Data Centre*. New York, NY: CRC/Auerbach, 2009, ch. Measurement, Metrics, and Management of IT resources.

- [14] J. Levandoski, P.-A. Larson, and R. Stoica, "Identifying hot and cold data in main-memory databases," in *Proc. of IEEE International Conference on Data Engineering*, April 2013, pp. 26–37.
- [15] D. Gibson, "Is your data hot, warm, or cold ?" 2012. [Online]. Available: <http://ibmdatamag.com/2012/06/is-your-big-data-hot-warm-or-cold/>
- [16] R. D. Strong, "Low-latency techniques for improving system energy efficiency," Ph.D. dissertation, University of California, San Diego, 2013.
- [17] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *IEEE Journal on Selected Areas in Communication*, May 2014.
- [18] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *IEEE Symposium on Mass Storage Systems and Technologies*, 2010, pp. 1–10.
- [19] M. Conway, "A multiprocessor system design," in *AFIPS Fall Joint Computing Conference*, 1963, pp. 139–146.
- [20] E. W. Dijkstra, "Cooperating sequential processes," in *Programming Languages*, 1968, pp. 43–112.
- [21] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," in *Information Theory Proceedings (ISIT)*, 2012, pp. 2766–2770.
- [22] N. B. Shah, K. Lee, and K. Ramchandran, "The MDS queue," *arXiv*, vol. abs/1211.5405, 2012.
- [23] N. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency ?" in *Allerton*, Oct 2013, pp. 731–738.
- [24] "How aws pricing works," July 2014. [Online]. Available: http://media.amazonwebservices.com/AWS_Pricing_Overview.pdf
- [25] "Google cloud storage - pricing." [Online]. Available: <https://cloud.google.com/storage/docs/storage-classes>
- [26] J. Plank, "Erasure codes for storage systems," Dec 2013. [Online]. Available: https://www.usenix.org/system/files/login/articles/10_plank-online.pdf
- [27] N. Cao, S. Yu, Z. Yang, W. Lou, and Y. Hou, "Lt codes-based secure and reliable cloud storage service," in *IEEE INFOCOM*, March 2012, pp. 693–701.
- [28] S. Aly, Z. Kong, and E. Soljanin, "Raptor codes based distributed storage algorithms for wireless sensor networks," in *IEEE International Symposium on Information Theory*, July 2008, pp. 2051–2055.
- [29] S. Chen, Y. Sun, U. Kozat, L. Huang, P. Sinha, G. Liang, X. Liu, and N. Shroff, "When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds," in *IEEE INFOCOM*, April 2014, pp. 1042–1050.
- [30] G. Liang and U. Kozat, "Use of erasure code for low latency cloud storage," in *Allerton*, Sept 2014, pp. 576–581.
- [31] —, "Fast cloud: Pushing the envelope on delay performance of cloud storage with coding," *IEEE/ACM Transactions on Networking*, vol. 22, no. 6, pp. 2012–2025, Dec 2014.
- [32] —, "Tofec: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes," in *Proc. of IEEE INFOCOM*, April 2014, pp. 826–834.
- [33] Y. Xiang, T. Lan, V. Aggarwal, and Y. F. R. Chen, "Joint latency and cost optimization for erasure-coded data center storage," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 2, pp. 3–14, Sep. 2014.
- [34] L. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec 2007.
- [35] D. Snowdon, S. Ruocco, and G. Heiser, "Power management and dynamic voltage scaling: Myths and facts," in *Proc. of Workshop on Power Aware Real-time Computing*, Sep 2005.
- [36] L. L. Andrew, M. Lin, and A. Wierman, "Optimality, fairness, and robustness in speed scaling designs," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 1, pp. 37–48, Jun. 2010.
- [37] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: Eliminating server idle power," *SIGARCH Comput. Archit. News*, vol. 37, no. 1, pp. 205–216, Mar. 2009.
- [38] Y. Liu, S. Draper, and N. S. Kim, "Queuing theoretic analysis of power-performance tradeoff in power-efficient computing," in *Conference on Information Sciences and Systems (CISS)*, March 2013, pp. 1–6.
- [39] —, "Sleepscale: Runtime joint speed scaling and sleep states management for power efficient data centers," in *IEEE International Symposium on Computer Architecture*, June 2014, pp. 313–324.
- [40] D. Bertsekas and R. Gallager, *Data Networks (2nd Ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1992, ch. Delay Models in Data Networks, pp. 203–206.
- [41] H. C. Tijms, *A first course in stochastic models*. Wiley, 2003.
- [42] E. Varki, A. Merchant, and H. Chen, "The M/M/1 fork-join queue with variable sub-tasks." [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.100.3062>
- [43] S. Ross, *A first course in probability*. Prentice-Hall Inc., 2002.
- [44] M. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: evidence and possible causes," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 835–846, Dec 1997.
- [45] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 251–262, Aug 1999.